More on “k-selection”

CSCI2100 Tutorial 4

Jianwen Zhao

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Introduction

Last week, in the lectures, we have learned the k-selection problem and solved it in $O(n)$ expected time by making use of randomization. The k-selection algorithm discussed in the class is easy to understand and analyze, but less efficient in practice.

In this tutorial, we will introduce a simpler and faster randomized algorithm (but with a more tedious analysis) for k-selection and do some exercises about this non-trivial problem.
A "simpler" randomized algorithm

1. Randomly pick an integer $v$ from $S$.
2. Get the rank of $v$, let it be $r$.
3. if $r = k$, return $v$, otherwise:
   3.1 if $r > k$, produce an array $S'$ containing all the integers of $S$ strictly smaller than $v$. Recurse on $S'$ by finding the $k$-th smallest element in $S'$.
   3.2 if $r < k$, produce an array $S'$ containing all the integers of $S$ strictly larger than $v$. Recurse on $S'$ by finding the $(k - r)$-th smallest element in $S'$.
Example

Consider that we want to find the 10-th smallest element from a set $S$ of 12 elements:

\[
17 \ 26 \ 38 \ 28 \ 41 \ 72 \ 83 \ 88 \ 5 \ 9 \ 12 \ 35
\]

Suppose that the $v$ we randomly choose is 28, whose rank is 6. Since $6 < 10$, we generate an array $S'$ with only the elements larger than 28:

\[
38 \ 41 \ 72 \ 83 \ 88 \ 35
\]

Then we can just recurse by finding the 4-th ($k - r = 10 - 6 = 4$) smallest element in this array $S'$. 
Cost Analysis

Clearly, \( f(1) = O(1) \). Consider \( n > 1 \).

Step 1 (on Slide 6) takes \( O(1) \) time.
Step 2 takes \( O(n) \) time.

Step 3:

(1) The rank \( r \) of the randomly selected integer \( v \) is uniformly distributed in \([1, n]\), namely, for each \( i \in [1, n] \), \( \Pr[r = i] = 1/n \).

(2) When \( r = i \), it determines a "left subset" \( S_l \) containing the \( i-1 \) integers of \( S \) smaller than \( v \), and a "right subset" \( S_r \) of size \( n - i \);

(3) In the worst case, we recurse into the larger one (\( \max\{i - 1, n - i\} \)) of these two subsets.
Let $X$ be the cost of Step 3,

$X$ equals to $f(\max\{1 - 1, n - 1\})$ with probability $1/n$. ($i = 1$)

$X$ equals to $f(\max\{2 - 1, n - 2\})$ with probability $1/n$. ($i = 2$)

$X$ equals to $f(\max\{3 - 1, n - 3\})$ with probability $1/n$. ($i = 3$)

\[ \cdots \]

$X$ equals to $f(\max\{n - 1, n - n\})$ with probability $1/n$. ($i = n$)

In general, $X$ equals to $f(\max\{i - 1, n - i\})$ with probability $1/n$.

Therefore, the expected cost of step 3 should be:

$$E[X] = \frac{1}{n} \cdot \sum_{i=1}^{n} f(\max\{i - 1, n - i\}) \quad (1)$$
Let $f(n)$ be the expected cost of our algorithm on an array of size $n$.

We know from the earlier analysis:

$$f(n) \leq \alpha \cdot n + \frac{1}{n} \cdot \sum_{i=1}^{n} f(\max\{i - 1, n - i\})$$

$$\leq \alpha \cdot n + \frac{2}{n} \cdot \sum_{i=\lceil n/2 \rceil}^{n} f(i - 1)$$

for some constant $\alpha > 0$.

We now prove that $f(n) = O(n)$ using substitution method.
Cost Analysis

We suppose that $f(n) \leq cn$ for some constant $c > 0$. First, this is obviously true for $n \leq 24$ when $c$ is at least a certain constant, say $\beta$ (when $n = O(1)$, the algorithm definitely finishes in constant time).

Assume that this holds for $n \leq k - 1$ for $k \geq 24$. Set $t = \lceil k/2 \rceil$. We have:

$$f(k) \leq \alpha \cdot k + \frac{2}{k} \cdot \sum_{i=t}^{k} c(i - 1)$$

$$\leq \alpha \cdot k + \frac{2c}{k} \cdot \frac{(t + k - 2)(k - t + 1)}{2}$$

$$< \alpha \cdot k + \frac{c(k^2 + 3k - t^2)}{k}$$

$$= (\alpha + c) \cdot k + 3c - \frac{ck}{k} \leq (\alpha + c) \cdot k + 3c - \frac{c(k/2)^2}{k}$$

$$= (\alpha + c) \cdot k + 3c - \frac{ck}{4}$$

(3)
Cost Analysis

From (3), we have:

\[ f(k) \leq (\alpha + c) \cdot k + 3c - \frac{ck}{4} \]  \hspace{1cm} (4)

We need the above to be at most \( ck \), namely:

\[ (\alpha + c) \cdot k + 3c - \frac{ck}{4} \leq ck \]

\[ \Leftrightarrow \alpha \cdot k + 3c \leq \frac{ck}{4} \]  \hspace{1cm} (5)

\[ \Leftrightarrow \begin{cases} 
ck/4 \geq 2\alpha k \\
ck/4 \geq 6c
\end{cases} \quad \Leftrightarrow \begin{cases} 
c \geq 8\alpha \\
k \geq 24
\end{cases} \]

Hence, setting \( c = \max\{8\alpha, \beta\} \) completes the proof.
Remark

The above algorithm is procedurally simpler than the one we taught in the class, and is faster in practice too. It, however, is less interesting in two ways:

1. Its analysis is more complicated (in the mundane way).
2. It does not illustrate the "if-failed-then-repeat" technique.
**Problem**

Let \( X[1\ldots n] \) and \( Y[1\ldots m] \) be two arrays, both sorted in ascending order. We want to find the \( k \)-th smallest of the \( n + m \) elements where \( 1 \leq k \leq n+m \). Our algorithm has to end in \( O(\log n + \log m) \) time.

**Example**

**Input**

\[
X: \begin{array}{cccccc}
2 & 3 & 6 & 7 & 9 & 12 \\
\end{array}
\quad Y: \begin{array}{cccccc}
1 & 4 & 8 & 10 & 11 \\
\end{array}
\quad k = 5
\]

**Output:** 6, since the final sorted array is:

\[
1 \ 2 \ 3 \ 4 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12
\]
Solution

We solve this problem by "resursion".

Base case:

The base case happens when either $n$ or $m$ is 1. Without loss of generality, assume that $m = 1$ (Otherwise, swap the roles of $X$ and $Y$).

- If $k = n + 1$, then return $\max\{X[n], Y[1]\}$.
- Otherwise (i.e., $k \leq n$):
  - If $X[k] < Y[1]$, then return $X[k]$.
  - Otherwise, return $\max\{X[k - 1], Y[1]\}$.

Obviously, the base case can be solved in $O(1)$ time.
**k-selection on two sorted arrays**

**Solution**

**Reduce case:**

Take:

1. The median element $u$ of $X$, namely, $u = X[s]$ where $s = \lfloor n/2 \rfloor$
2. The median element $v$ of $Y$, namely, $v = Y[t]$ where $t = \lfloor m/2 \rfloor$

Without loss of generality, we assume $v \leq u$ (Otherwise, swap the roles of $X$ and $Y$). We distinguish two cases:

- **Case 1:** $s + t \geq k$: None of the elements in $X[s + 1, \ldots n]$ can possibly be the result. We recurse by searching for the $k$-th smallest element of the $s + m$ elements in $X[1\ldots s]$ and $Y[1\ldots m]$.

- **Case 2:** $s + t < k$: None of the elements in $Y[1, \ldots t]$ can possibly be the result. We recurse by searching for the $(k - t)$-th smallest element of the $n + m - t$ elements in $X[1\ldots n]$ and $Y[t + 1\ldots m]$.
k-selection on two sorted arrays

Example

Input $X$: 2 8 11 17 20 33 35  
$Y$: 1 4 7 28 30 43  $k = 5$

Where $n = 7$, $m = 6$, $s = \lfloor n/2 \rfloor = 3$, $t = \lfloor m/2 \rfloor = 3$.

We take $u = X[s] = 11$, $v = Y[t] = 7$, and $u > v$.

Since $k = 5$, $s + t = 6 > k$, which follows case 1, then none of the elements in $X[4, ... 7]$ can possibly be the result. We recurse by searching for the 5-th smallest element of the 9 elements in $X[1...3]$ and $Y[1...6]$, i.e.

New Input 1  
$Y$: 2 8 11  
$X$: 1 4 7 28 30 43  $k = 5$
**k-selection on two sorted arrays**

**Example**

New Input 1

\[ Y: \begin{bmatrix} 2 & 8 & 11 \end{bmatrix} \quad X: \begin{bmatrix} 1 & 4 & 7 & 28 & 30 & 43 \end{bmatrix} \quad k = 5 \]

Where \( n = 6, m = 3, s = \lfloor n/2 \rfloor = 3, t = \lfloor m/2 \rfloor = 1. \)

We take \( u = X[s] = 7, v = Y[t] = 2, \) and \( u > v. \)

Since \( k = 5, s + t = 4 < k, \) which follows case 2, then \( Y[1] \) cannot possibly be the result. We recurse by searching for the \( 5 - 1 = 4 \)-th smallest element of the 8 elements in \( X[1...6] \) and \( Y[2...3], \) i.e.

New Input 2

\[ X: \begin{bmatrix} 8 & 11 \end{bmatrix} \quad Y: \begin{bmatrix} 1 & 4 & 7 & 28 & 30 & 43 \end{bmatrix} \quad k = 4 \]
Example

New Input 2  
\[X: 8 \quad 11 \quad Y: 1 \quad 4 \quad 7 \quad 28 \quad 30 \quad 43 \quad k = 4\]

Where \(n = 2\), \(m = 6\), \(s = \lfloor n/2 \rfloor = 1\), \(t = \lfloor m/2 \rfloor = 3\).

We take \(u = X[s] = 8\), \(v = Y[t] = 7\), and \(u > v\).

Since \(k = 4\), \(s + t = 4 = k\), which follows case 1, then \(X[2]\) cannot possibly be the result. We recurse by searching for the 4-th smallest element of the 7 elements in \(X[1]\) and \(Y[1...6]\), i.e.

New Input 3  
\[Y: 8 \quad X: 1 \quad 4 \quad 7 \quad 28 \quad 30 \quad 43 \quad k = 4\]
Example

New Input 3 \( Y: 8 \quad X: \begin{array}{cccccc} 1 & 4 & 7 & 28 & 30 & 43 \end{array} \quad k = 4 \)

This comes to be the base case, since \( k = 4 < n = 6 \),
\( X[k] = X[4] = 28 > Y[1] \), we return \( \max\{X[k-1], Y[1]\} = 8 \).
k-selection on two sorted arrays

Cost Analysis

From the above example, we can see that for each recursion, we shrink either $X$ or $Y$ by half. Overall, the above shrinking can happen at most $\log_2 m + \log_2 n$ times before reaching the base case.

It thus follows that the entire algorithm finishes in $O(\log n + \log m)$ time.